第2章

データ・バッファやCPUコアに不可欠なメモリを使いこなす

――同期メモリと非同期メモリの違いを理解する

山口耕作

ここでは，FPGAに搭載されたメモリの活用法を解説する．通信やディジタル信号処理のようなアプリケーションではデータ・バッファとして，組み込みシステムなどのアプリケーションではCPUコアのための命令/データ・メモリとして，メモリは不可欠である．そこで最近のFPGAは，大容量のメモリ・ブロックを搭載し，さまざまな構成で使えるようになっている．（編集部）

FPGA（field programmable gate array）の基本特許であるFreeman特許の要点の一つに，「メモリを利用したLUT（look-up table）で論理回路を構成する」というものがあります．そもそも，FPGAと内蔵メモリは切っても切り離せない関係にあるといえます．

米国Xilinx社のFPGAの場合，1998年に発売されたVirtexシリーズからBlock RAMというハード・マクロのデュアルポート・メモリ・ブロックが内蔵されるようになりました．FPGA内蔵メモリの用途としては，まず一般のROM/RAMと同じくデータの格納に使用されることは当然なのですが，このほかに，Virtexファミリのメモリは，高速通信のためのFIFOバッファなどとして使いやすいようになっていました．また，最近ではFPGAにおけるCPUコアの活用が注目されていますが，FPGA内蔵メモリはCPUのための命令メモリやデータ・メモリとしても重要です．

FPGAに集積されるメモリ容量は，増加の一途をたどっており，低コストFPGAファミリであっても大きな容量のメモリを持つようになりました．

FPGA内蔵メモリの特徴

Xilinx社のSpartan-3ファミリを例に，FPGAが持つメモリの特徴について説明します．

Spartan-3は，同期書き込み（**図1(a)**）と同期読み出し（**図1(b)**）のメモリ・ブロック（Block RAM）と，同期書き込みと非同期読み出し（**図1(c)**）の分散メモリ（Distributed Memory）の2種類があります．

● メモリ・ブロック

Spartan-3のBlock RAMは，1ブロック当たり18Kビットの容量を持ちます．従来の低価格FPGAであるSpartan-

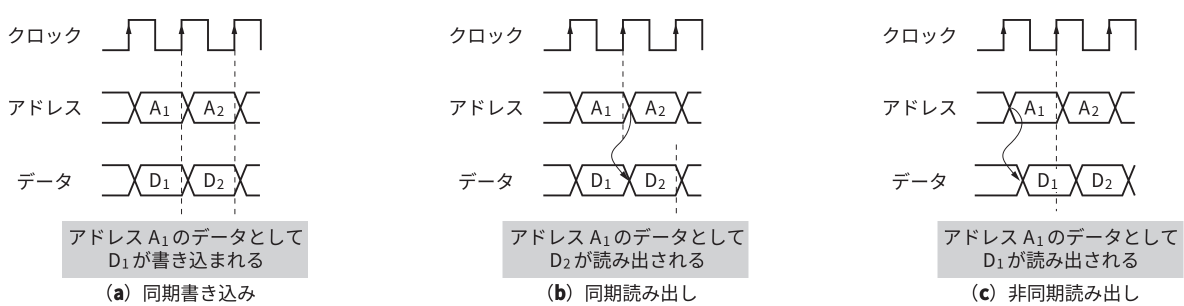


図1 メモリ・アクセスのタイミング
Spartan-3は，同期書き込みと同期読み出しのメモリ・ブロック（Block RAM）と，同期書き込みと非同期読み出しの分散メモリを持つ．

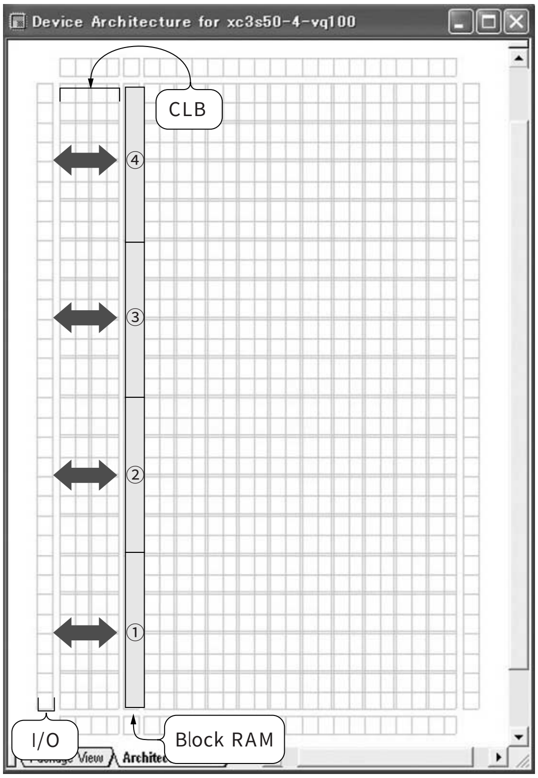


図2　Spartan-3が持つメモリ・ブロック
Spartan-3のうち，5万システム・ゲート品のXC3S50VQ100-4の例を示す．四つのメモリ・ブロックがあり，それらと外部I/Oの間にCLB（Configurable Logic Block）がある．メモリ上の比較的大容量のデータを，最小限の論理回路だけを使って入出力するのに適した構成といえる．

II/IIEでは4Kビットのブロックでした．本誌2005年1月号に付属していたXC3S50は，4ブロックのBlock RAMを持ちます．したがって，合計で72Kビットです（**図2**）．

メモリ・ブロックでは，すべてのメモリ領域を自由に使えるわけではありません．例えばSpartan-3のBlock RAMでは，実際にデータとして使用できるのは1ブロック当たり16Kビットです．残りの2Kビットはメモリ内容の信頼性向上のためのパリティ・ビットとして使用されます．

● 分散メモリ

分散メモリは，基本論理ブロックを構成する4入力LUTを本来のメモリの用途で使います．4ビットのアドレスと1ビットのデータを持つエレメントが基本になります．複数のエレメントを組み合わせることで，任意のデータ幅やアドレス幅に拡張して使用できます．

メモリの使いかた

メモリを使う設計には，大きく3通りの方法があります．

● 汎用的なHDL記述

特定のFPGAやASICに依存しない，汎用的なメモリの

リスト1　メモリの汎用的なVHDL記述

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;        ← conv_integerを使用するためのパッケージ呼び出し

entity memtest is
    generic(
        DATA_WIDTH : integer := 8;      ← データ幅の定義
        ADDR_WIDTH : integer := 11);    ← アドレス幅の定義
    Port (
        clk  : in  std_logic;
        a    : in  std_logic_vector(ADDR_WIDTH-1 downto 0);
        di   : in  std_logic_vector(DATA_WIDTH-1 downto 0);
        do0  : out std_logic_vector(DATA_WIDTH-1 downto 0);
        do1  : out std_logic_vector(DATA_WIDTH-1 downto 0);
        we   : in  std_logic_vector(1 downto 0));
end memtest;

architecture rtl of memtest is

    constant DIST_ADDR_WIDTH : integer := 5;    ← メモリのアドレス幅（非同期メモリ用）
    constant BRAM_ADDR_WIDTH : integer := 11;   ← メモリのアドレス幅（同期メモリ用）

    type distram_type is array (2**DIST_ADDR_WIDTH-1 downto 0)
          of std_logic_vector (DATA_WIDTH-1 downto 0);      ①
    signal dist_spram0 : distram_type;                      ②
                                                 ← メモリの定義（非同期メモリ用）

    type bram_type is array (2**BRAM_ADDR_WIDTH-1 downto 0) of
                     std_logic_vector (DATA_WIDTH-1 downto 0);
    signal bram_spram0 : bram_type;
                                                 ← メモリの定義（同期メモリ用）
begin

-- Distributed RAM by HDL ------------------------
dist_spram0_inst : process (clk)
begin
    if (clk'event and clk = '1') then
        if (we(0) = '1') then
            dist_spram0(conv_integer(a(DIST_ADDR_
                      WIDTH-1 downto 0))) <= di;   ③
        end if;
    end if;
end process;

do0 <= dist_spram0(conv_integer(a(DIST_ADDR_WIDTH-
                                  1 downto 0)));
                 非同期メモリの記述→分散メモリを使用
-- BRAM16 with write first mode by HDL -----------
bram_spram0_inst : process (clk)
begin
    if (clk'event and clk = '1') then
        if (we(1) = '1') then
            bram_spram0(conv_integer(a(BRAM_ADDR_
                      WIDTH-1 downto 0))) <= di;   ③
            do1 <= di;
        else
            do1 <= bram_spram0(conv_integer(a(BRAM_
                           ADDR_WIDTH-1 downto 0)));
        end if;
    end if;
end process;

end rtl;      同期メモリの記述→メモリ・ブロックを使用
```

VHDLによる記述例を**リスト1**に示します．ここでは，データ幅を8ビット（DATA_WIDTHで指定）とし，アドレス幅5ビット（DIST_ADDR_WIDTHで指定）で32バイト（256ビット）の非同期メモリと，アドレス幅11ビット（BRAM_ADDR_WIDTHで指定）で2,048バイト（16Kビット）の同期メモリを生成しています．

メモリ特有の記述について解説します．メモリの記述は，C言語の構造体宣言のように，まず型宣言を行ってからその型を実体に割り当てるようにします．書式を**図3**に示します．**リスト1**の①は，いくらのデータ幅のメモリ要素をいくつ作るかを指定する型の宣言です．宣言には配列タイプを示すarrayを使います．**リスト1**の②はメモリの実体（インスタンス）の定義です．タイプ名が特殊なだけで通常のsignal文と同じです．

以上でデータをためる入れ物は準備できました．ここまでは同期読み出しメモリも非同期読み出しメモリもまったく同じです．実際にメモリとして使用するには，この入れ物にどのようにしてデータを書き込み，どのようにして読み出すかを指定する必要があります．この指定で同期/非同期が決まります．

リスト1の③では指定番地のメモリにdiのデータを書き込んでいます．代入演算子（<=）の左辺はややこしい記述になっていますが，わかりやすくすると**図4**のようになります．つまり，「アドレス・バス信号の値を整数変換して，その順番の要素を指定する」ということです．そして，メモリから読み書きするコードを，クロックをセンシティビティ・リストに持つprocess文内（もっと具体的にはif (clk'event and clk = '1') thenとend if;の中）に入れれば同期メモリとなり，process文の外に置けば非同期メモリとなります．これはどのような回路でも同じです．

なお，std_logic_vectorを整数に変換するconv_integerを使うためには，パッケージSTD_LOGIC_UNSIGNEDを呼び出す必要があります（**リスト1**の4行目）．実はSTD_LOGIC_UNSIGNED内でSTD_LOGIC_1164とSTD_LOGIC_ARITHが呼び出されているので，STD_LOGIC_UNSIGNEDさえあればあとの二つがなくても**リスト1**はエラーなくコンパイル（論理合成・配置配線）できてしまいます．

Xilinx社のXC3S50VQ100-4とAltera社のEP1S10F780C5ESをターゲットとしてコンパイルした場合の結果を**図5**に示します．非同期メモリは基本論理ブロック部分に，同期メモリはハード・マクロのメモリに割り当てられることがわかります．これは，ISEでは拡張子.mrp，Quartus IIでは拡張子.map.rptのファイルで確認できます．

非同期メモリとしては，16個のLUTが使用されています．一つの分散メモリ（32x1 RAM）は2個のLUTで構成されるので，8個の分散メモリに割り当てられたことになります．フロアプランナで表示されるリソースのリストでも確認できます．

● エレメント名を直接指定する

次に，デバイス特有のエレメント名を直接指定してこれらのメモリを実現するVHDL記述を**リスト2**に示します．デバイスに依存するので，ここではXilinx社のSpartan-3向けの例を説明します．

エレメント名を直接指定する場合は，まずデバイスに特有のライブラリとパッケージを指定する必要があります．それが**リスト2**の6～7行目にある以下の記述です．

2

```
type<タイプ名>is array(<データ個数>-1 downto 0)of std _logic_vector(<データ幅>-1 downto 0);
                 ↑ 何個作るかの指定                        ↑ 1個当たりのデータ幅の指定

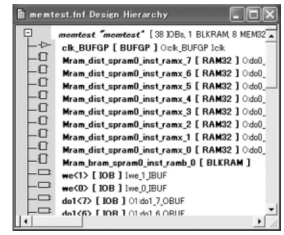
signal<メモリ名>:<タイプ名>;
```

図3 メモリの定義
型宣言を行ってからその型を実体に割り当てる．

```
dist_spram0(                                      )←― 定義したメモリ名
          conv_integer(                          )←― std_logic_vectorを整数に変換
                    a(DIST_ADDR_WIDTH-1 downto 0)←― アドレス・バス信号
```

図4 指定番地のメモリの記述
アドレス・バス信号の値を整数変換して，その順番の要素を指定する．

```
Logic Distribution:
  Number of occupied Slices:                            8 out of     768    1%
    Number of Slices containing only related logic:       8 out of       8  100%
    Number of Slices containing unrelated logic:          0 out of       8    0%
      *See NOTES below for an explanation of the effects of unrelated logic
Total Number 4 input LUTs:              16 out of   1,536    1%
  Number used for 32x1 RAMs:             16 <------------------- 非同期メモリ
    (Two LUTs used per 32x1 RAM)
  Number of bonded IOBs:                38 out of      63   60%
  Number of Block RAMs:                 1 out of        4   25% <-- 同期メモリ
  Number of GCLKs:                      1 out of        8   12%
```



(**a**) Spartan-3 (XC3S50VQ100-4) の場合

```
+-------------------------------------+-------------+--------------+-------------+-----------
; Compilation Hierarchy Node          ; Logic Cells ; LC Registers ; Memory Bits ; DSP Ele...
+-------------------------------------+-------------+--------------+-------------+-----------
; |memtest                            ; 508 (304)   ; 256          ; 16384       ; 0
;    |altsyncram:bram_spram0_rtl_0|   ; 0 (0)       ; 0            ; 16384       ; 0
;       |altsyncram_h2t:auto_generated| ; 0 (0)     ; 0            ; 16384       ; 0
;    |lpm_mux:dist_spram0_rtl_1|      ; 28 (0)      ; 0            ; 0           ; 0
;       |mux_2ic:auto_generated|      ; 28 (28)     ; 0            ; 0           ; 0
;    |lpm_mux:dist_spram0_rtl_2|      ; 24 (0)      ; 0            ; 0           ; 0
;       |mux_2ic:auto_generated|      ; 24 (24)     ; 0            ; 0           ; 0
;    |lpm_mux:dist_spram0_rtl_3|      ; 24 (0)      ; 0            ; 0           ; 0
;       |mux_2ic:auto_generated|      ; 24 (24)     ; 0            ; 0           ; 0
;    |lpm_mux:dist_spram0_rtl_4|      ; 24 (0)      ; 0            ; 0           ; 0
;       |mux_2ic:auto_generated|      ; 24 (24)     ; 0            ; 0           ; 0
;    |lpm_mux:dist_spram0_rtl_5|      ; 24 (0)      ; 0            ; 0           ; 0
;       |mux_2ic:auto_generated|      ; 24 (24)     ; 0            ; 0           ; 0
;    |lpm_mux:dist_spram0_rtl_6|      ; 28 (0)      ; 0            ; 0           ; 0
;       |mux_2ic:auto_generated|      ; 28 (28)     ; 0            ; 0           ; 0
;    |lpm_mux:dist_spram0_rtl_7|      ; 26 (0)      ; 0            ; 0           ; 0
;       |mux_2ic:auto_generated|      ; 26 (26)     ; 0            ; 0           ; 0
;    |lpm_mux:dist_spram0_rtl_8|      ; 26 (0)      ; 0            ; 0           ; 0
;       |mux_2ic:auto_generated|      ; 26 (26)     ; 0            ; 0           ; 0
+-------------------------------------+-------------+--------------+-------------+-----------
```

同期メモリ（Memory Bits 16384）

非同期メモリ（lpm_mux:dist_spram0_rtl_1～8）

(**b**) Stratix (EP1S10F780C5ES) の場合

図5 コンパイル結果

非同期メモリは基本論理ブロック部分に，同期メモリはハード・マクロのメモリに割り当てられる．Spartan-3のBlock RAMは1個だけ使われている．Stratixでは，インスタンス名を見れば，同期メモリのbram_spram0はメモリに，非同期メモリのdist_spram0は基本論理ブロックに割り当てられていることがわかる．なお，コンパイル結果はツールのバージョンやオプション設定によって変わることがある．

リスト2 明示的にメモリのエレメントを指定

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

library UNISIM;
use UNISIM.VComponents.all;

entity memtest is
    generic(
        DATA_WIDTH : integer := 8;
        ADDR_WIDTH : integer := 11);
    Port (
        clk : in  std_logic;
        a   : in  std_logic_vector(ADDR_WIDTH-1 downto 0);
        di  : in  std_logic_vector(DATA_WIDTH-1 downto 0);
        do0 : out std_logic_vector(DATA_WIDTH-1 downto 0);
        do1 : out std_logic_vector(DATA_WIDTH-1 downto 0);
        we  : in  std_logic_vector(1 downto 0));
end memtest;

architecture rtl of memtest is

    signal net_gnd1, net_vcc1 : std_logic_vector
                                          (0 downto 0);

begin

    net_vcc1 <= "1";
    net_gnd1 <= "0";

-- Distributed RAM by Premitive
dist_spram1_inst0 : RAM16X4S
   generic map (
      INIT_00 => X"0000", --  INIT for bit 0 of RAM
      INIT_01 => X"0000", --  INIT for bit 1 of RAM
      INIT_02 => X"0000", --  INIT for bit 2 of RAM
      INIT_03 => X"0000") --  INIT for bit 3 of RAM
   port map (
      O0 => do0(0),      -- RAM data[0] output
      O1 => do0(1),      -- RAM data[1] output
      O2 => do0(2),      -- RAM data[2] output
      O3 => do0(3),      -- RAM data[3] output
      A0 => a(0),        -- RAM address[0] input
      A1 => a(1),        -- RAM address[1] input
      A2 => a(2),        -- RAM address[2] input
      A3 => a(3),        -- RAM address[3] input
      D0 => di(0),       -- RAM data[0] input
      D1 => di(1),       -- RAM data[1] input
      D2 => di(2),       -- RAM data[2] input
      D3 => di(3),       -- RAM data[3] input
      WCLK => clk,       -- Write clock input
      WE => we(0)        -- Write enable input
   );

dist_spram1_inst1 : RAM16X4S
   generic map (
      INIT_00 => X"0000", --  INIT for bit 0 of RAM
      INIT_01 => X"0000", --  INIT for bit 1 of RAM
      INIT_02 => X"0000", --  INIT for bit 2 of RAM
      INIT_03 => X"0000") --  INIT for bit 3 of RAM
   port map (
      O0 => do0(4),      -- RAM data[0] output
      O1 => do0(5),      -- RAM data[1] output
      O2 => do0(6),      -- RAM data[2] output
      O3 => do0(7),      -- RAM data[3] output
      A0 => a(0),        -- RAM address[0] input
      A1 => a(1),        -- RAM address[1] input
      A2 => a(2),        -- RAM address[2] input
      A3 => a(3),        -- RAM address[3] input
```

分散メモリ＝非同期メモリ 2個の4ビット・エレメントを使用して8ビット幅にしている

1エレメント目の初期値の定義

2エレメント目の初期値の定義

```
library UNISIM;
use UNISIM.VComponents.all;
```

Spartan-3では，分散メモリのエレメント名は「RAM16X4S」，Block RAMのエレメント名は「RAMB16_S9」です．18Kビットのブロックですが，2Kビットがパリティ用なので「RAMB18」ではなく「RAMB16」となることに注意が必要です．対象デバイスでどのようなエレメントが使えるのかを調べるには，オンライン・マニュアルのライブラリ・ガイドを参照します(**図6**).

この記述方法はVHDLソース・コードでメモリ内容の初期値を指定できるので便利ですが，記述量が非常に多くなってしまいます．ここで記述のサンプル集であるLanguage Templatesを使えば，比較的簡単に記述できます[(1)]．ISEの「Edit」→「Language Templates」に記述例があるので，これをコピー・アンド・ペーストし，インスタンス名を決

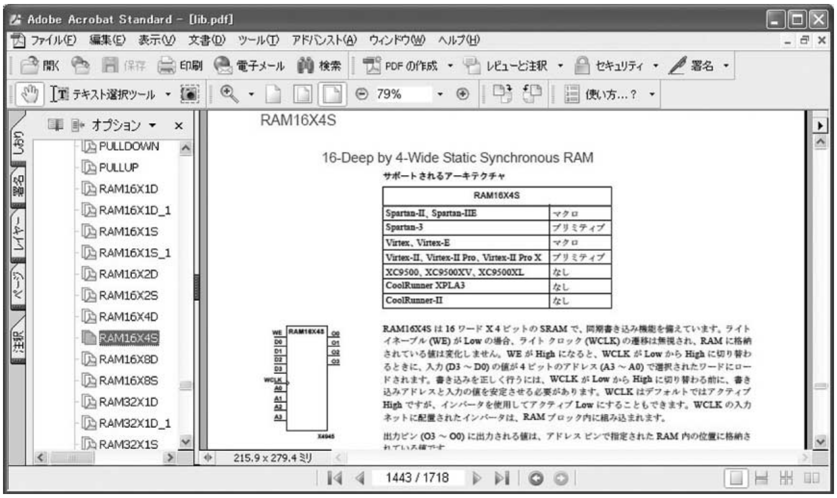


図6
ライブラリ・ガイド
RAM16X4SはSpartan-3でプリミティブとして使用可能．


```
        D0 => di(4),      -- RAM data[0] input
        D1 => di(5),      -- RAM data[1] input
        D2 => di(6),      -- RAM data[2] input
        D3 => di(7),      -- RAM data[3] input
        WCLK => clk,      -- Write clock input
        WE => we(0)       -- Write enable input
    );

-- BRAM16 by Premitive ---------------
bram_spram1_inst : RAMB16_S9
    generic map (
        INIT => X"000",  --  Value of output RAM registers at startup
        SRVAL => X"000", --  Ouput value upon SSR assertion
        WRITE_MODE => "WRITE_FIRST", --  WRITE_FIRST, READ_FIRST
                                              or NO_CHANGE
        INIT_00 => X"0000000000000000000000000000000000000000000
                                    000000000000000000000",
        INIT_01 => X"0000000000000000000000000000000000000000000
                                    000000000000000000000",
        INIT_02 => X"0000000000000000000000000000000000000000000
                                    000000000000000000000",
        INIT_03 => X"0000000000000000000000000000000000000000000
                                    000000000000000000000",

         (中略)

        INIT_3D => X"0000000000000000000000000000000000000000000
                                    000000000000000000000",
        INIT_3E => X"0000000000000000000000000000000000000000000
                                    000000000000000000000",
        INIT_3F => X"0000000000000000000000000000000000000000000
                                    000000000000000000000",
        INITP_00 => X"000000000000000000000000000000000000000000
                                    0000000000000000000000",
        INITP_01 => X"00000000000000000000000000000000000000000
                                 00000000000000000000000",
        INITP_02 => X"0000000000000000000000000000000
                     000000000000000000000000000000000",
        INITP_03 => X"0000000000000000000000000000000
                     000000000000000000000000000000000",
        INITP_04 => X"0000000000000000000000000000000
                     000000000000000000000000000000000",
        INITP_05 => X"0000000000000000000000000000000
                     000000000000000000000000000000000",
        INITP_06 => X"0000000000000000000000000000000
                     000000000000000000000000000000000",
        INITP_07 => X"0000000000000000000000000000000
                     000000000000000000000000000000000")
    port map (
        DO => do1,              -- 8-bit Data Output
        DOP => open,            -- 1-bit parity Output
        ADDR => a(10 downto 0), -- 11-bit Address Input
        CLK => clk,             -- Clock
        DI => di,               -- 8-bit Data Input
        DIP => net_gnd1(0 downto 0), -- 1-bit parity
                                        Input
        EN => net_vcc1(0),      -- RAM Enable Input
        SSR => net_gnd1(0),     -- Synchronous
                                      Set/Reset Input
        WE => we(1)             -- Write Enable Input
    );

end rtl;
```

ブロック・メモリ＝同期メモリ

データ初期値

パリティ初期値

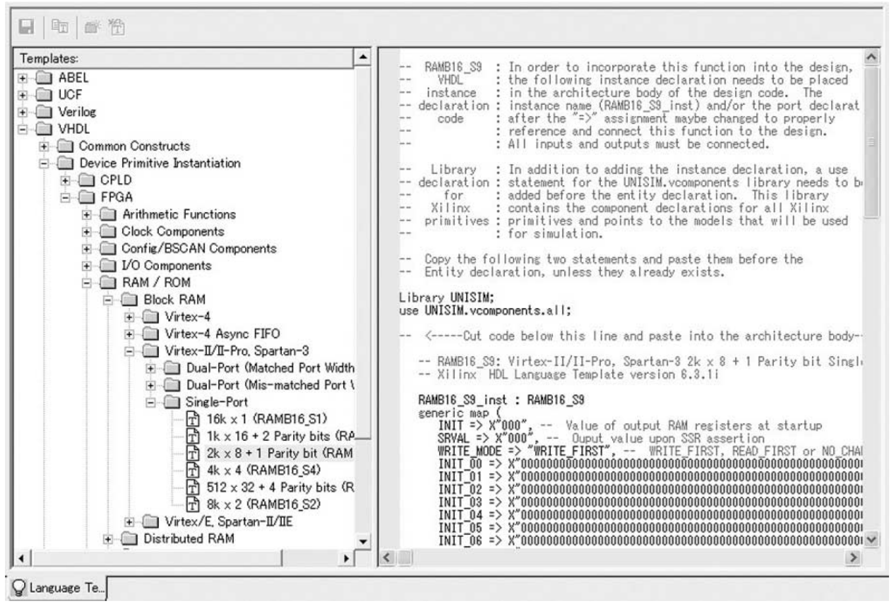


図7
Language Templates
RAMB16_S9の例．これをコピー・アンド・ペーストする．

めて port map 部分に所望の信号を接続します(**図7**)．

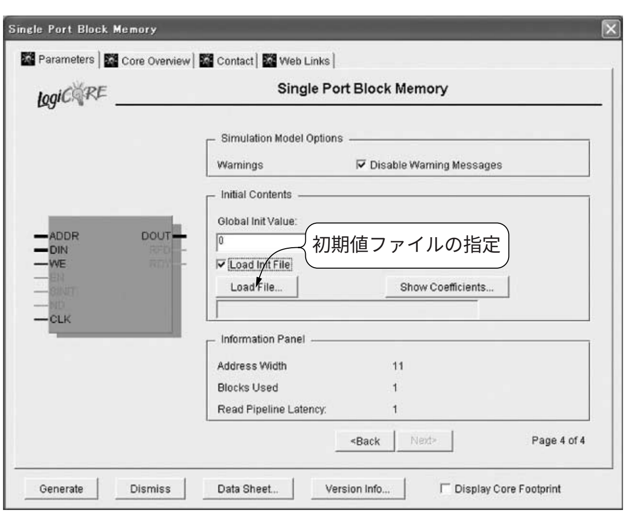
Spartan-3の分散メモリを利用する場合，注意しなければならないことがあります．それは，ほかのFPGAがサポートしていた分散メモリのマクロを使えない場合があることです．例えば，後ほど紹介する設計例では，「RAM16X4S」を二つ組み合わせて8ビット・データ幅の分散メモリを作っています．これはSpartan-IIEなどでは指定可能な8ビット・データ幅のマクロ「RAM16X8S」がSpartan-3では指定できないからです．

● IPコア生成ツールでカスタマイズ

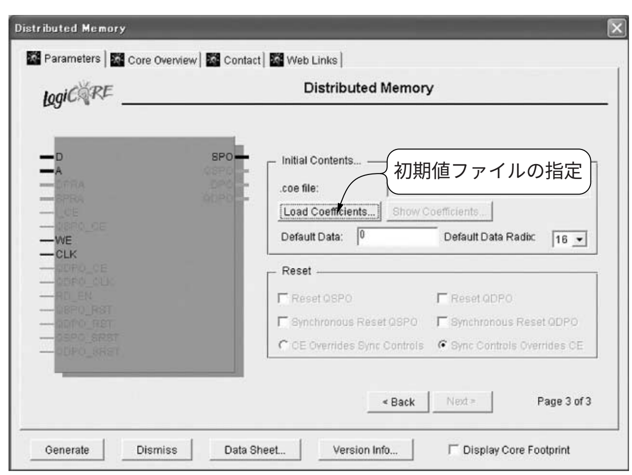
また，有償版のISEでは，CORE GeneratorというIPコア生成ツールを使用できます．これを使えば，メモリの構成をGUIで設定できるようになります．また初期値を別ファイルで指定することもできます(**図8**)．

このCORE Generatorを使って生成した，メモリを呼び出すVHDL記述を**リスト3**に示します．今までの中でもっともシンプルなコードになっていることがわかると思います．

CORE Generatorは，ユーザが生成したメモリを呼び出す方法をテンプレートとして，拡張子.vhoのファイルをプ



(**a**) Block RAMの生成



(**b**) 分散メモリの生成

図8　CORE Generatorの画面

リスト3　CORE Generatorで生成したメモリの呼び出し

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

library UNISIM;
use UNISIM.VComponents.all;

entity memtest is
    generic(
        DATA_WIDTH : integer := 8;
        ADDR_WIDTH : integer := 11);
    Port (
        clk : in  std_logic;
        a   : in  std_logic_vector(ADDR_WIDTH-1 downto 0);
        di  : in  std_logic_vector(DATA_WIDTH-1 downto 0);
        do0 : out std_logic_vector(DATA_WIDTH-1 downto 0);
        do1 : out std_logic_vector(DATA_WIDTH-1 downto 0);
        we  : in  std_logic_vector(1 downto 0));
end memtest;

architecture rtl of memtest is

component distspram2
    port (
    A:   IN  std_logic_VECTOR(3 downto 0);
    CLK: IN  std_logic;
    D:   IN  std_logic_VECTOR(7 downto 0);
    WE:  IN  std_logic;
    SPO: OUT std_logic_VECTOR(7 downto 0));
end component;

component bramspram2
    port (
    addr: IN  std_logic_VECTOR(10 downto 0);
    clk:  IN  std_logic;
    din:  IN  std_logic_VECTOR(7 downto 0);
    dout: OUT std_logic_VECTOR(7 downto 0);
    we:   IN  std_logic);
end component;

begin
-- Distributed RAM by CoreGen---------------------
dist_spram2 : distspram2
        port map (
            A => a(3 downto 0),
            CLK => clk,
            D => di,
            WE => we(0),
            SPO => do0);

-- BRAM16 by CoreGen -----------------------------
bram_spram2 : bramspram2
        port map (
            addr => a,
            clk => clk,
            din => di,
            dout => do1,
            we => we(1));

end rtl;
```

ロジェクト・フォルダに生成します．その内容をコピー・アンド・ペーストすれば，基本の記述はすぐに完成です．

メモリを使う設計のテクニック

メモリを使う設計を行う際に知っておくべきテクニックを紹介します．

● メモリを意識した設計が必要

Xilinx社のSpartan-3などには，メモリ・ブロックと分散メモリがあると説明しましたが，その使い分けはどのようにしたらよいのでしょうか？

分散メモリは，数バイト程度のわずかなメモリ容量だけが必要な場合，もしくは非同期読み出しがどうしても必要な場合に，効率的にメモリ機能を実現します．しかし，逆に言えばこれらの場合以外は，論理ブロックを浪費しないためにも，メモリ機能はできるだけメモリ・ブロックを使うほうが賢明だと筆者は考えます．

メモリ・ブロックや分散メモリといったことをあまり意識せず，FPGA設計ツールに任せてしまうことはできるのでしょうか？

ブロック・メモリと分散メモリでは，その構造やアクセス・タイミングが根本的に違うため，どちらかがリソース不足になったときに互いにカバーし合うようなことは，残念ながら自動ではやってくれません．結局，設計者が意識的にメモリの機能と容量などを管理する必要があります．

● 1本の信号の違いで回路規模が大きく異なる

メモリでは同期読み出しか非同期読み出しかの違いによって，ブロック・メモリが使われるか分散メモリが使われるかが決定されます．このため，たった1本の信号の違いで回路規模が大きく異なる場合があります．

表1はシフト・レジスタにおいてリセットがある場合とない場合の比較を示しています．

リセットがないシフト・レジスタは，SRL16というLUTを使った回路が適用されるため，フリップフロップの数を極端に減らすことができます．回路規模が小さいぶん，性能も向上します．

リセットが入力されたときに回路全体を初期化するというのは，設計の基本です．しかし，FPGAのリソースが不足してしまった場合やチューニングを行う場合などには，ほんとうにそのシフト・レジスタにリセットが必要かどうかを判断して，SRL16を有効活用してください．

表1 リセットの有無で回路規模が大きく異なるシフト・レジスタ

リセット	なし	あり
VHDL記述	**process** (clk) **begin** **if** clk'event **and** clk = '1' **then** **if** clk_en = '1' **then** reg0 <= reg0(6 **downto** 0) & di; **end if**; **end if**; **end process**; **do** <= reg0(7);	**process** (clk, rst_n) **begin** **if** rst_n = '0' **then** reg0 <= (others =>'0'); **elsif** clk'event **and** clk = '1' **then** **if** clk_en = '1' **then** reg0 <= reg0(6 **downto** 0) & di; **end if**; **end if**; **end process**; **do** <= reg0(7);
使用リソース	Slice数＝1 フリップフロップ数＝1	Slice数＝3 フリップフロップ数＝8

```
data2mem
    -bm〈配置情報付きBMMファイル名.bmm〉
    -bd〈メモリ・ファイル名.mem〉tag〈BMMファイル内のタグ名〉
    -bd〈メモリ・ファイル名.mem〉tag〈BMMファイル内のタグ名〉

    ......

    -bt〈入力ビットストリーム名.bit〉
    -ob〈出力ビットストリーム名.bit〉
```

図9 DATA2MEMによるメモリ初期化に必要なオプション

● 初期値の指定

エレメント名を明示的に記述する方法やCORE Generatorを使う方法ではメモリの初期値を指定できることを説明しました．しかし，いずれの場合も，初期値の内容に少しでも変更があったときは，コンパイルをやり直す必要があります．大規模な設計では，コンパイル時間がかなりかかります．たった1バイトのメモリ変更のために再コンパイルということはできれば避けたいものです．

Xilinx社のISEの場合，FPGAをコンフィグレーションするビットストリーム・データのメモリの初期値の内容を直接変更できるDATA2MEMというツールがあります．つまり，データの変更があっても，論理合成，配置配線をまったく変更せずに，メモリ内容だけを変更できるということです．この機能は，Xilinx社のEmbedded Development Kit（EDK）を使ってCPUコアを用いた設計を行っている人であればよく知っていると思います．そして，DATA2MEMそのものはEDKではなくISEの機能なので，CPUコアの利用にかかわらず活用できます．

DATA2MEMは便利な反面，細かいファイル設定が必要になります．そういう意味では，使うには少々テクニックが必要です．**図9**にコマンド実行に必要なオプションを示します．

リスト4 配置情報付きBMMファイルの例

```
ADDRESS_SPACE tag1 RAMB16 [0x00000000:0x000007FF]
/* "tag1"というのが「タグ名」 */
    BUS_BLOCK
        bram_spram1_inst [7:0] PLACED = X0Y1;
  /* インスタンス名とともにブロック・メモリの配置情報を記述 */
    END_BUS_BLOCK;
END_ADDRESS_SPACE;
```

リスト5 メモリ・ファイル例

```
@0                           /* @付きでアドレスを指定する */
00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f
                             /* あとはデータを列挙するだけ */
1f 1e 1d 1c 1b 1a 19 18 17 16 15 14 13 12 11 10
```

リスト4と**リスト5**にDATA2MEMに入力するファイルの内容の例を示します．タグとメモリ・ファイルは1対1で対応するので，一つのFPGA内に複数の独立したブロック・メモリがある場合，それぞれに別々の初期値を与えることが可能です．

参考・引用*文献

(1) 山口耕作；Xilinxデバイス活用時のトラブルと解決法① デバイス内蔵機能の使いかたがわからない，Design Wave Magazine，p.58，2002年3月号．

株式会社ダイヘン
やまぐち・こうさく